CFU Playground

Custom
Function
Unit

Build your own ML Processor
using Open Source and FPGAs

WOSET -- Workshop on Open Source EDA Technologies
Autumn, 2021

Tim Callahan, tcal@google.com
Tim Ansell, Joey Bushagour, Alan Green, Dan Callaghan, David Lattimore

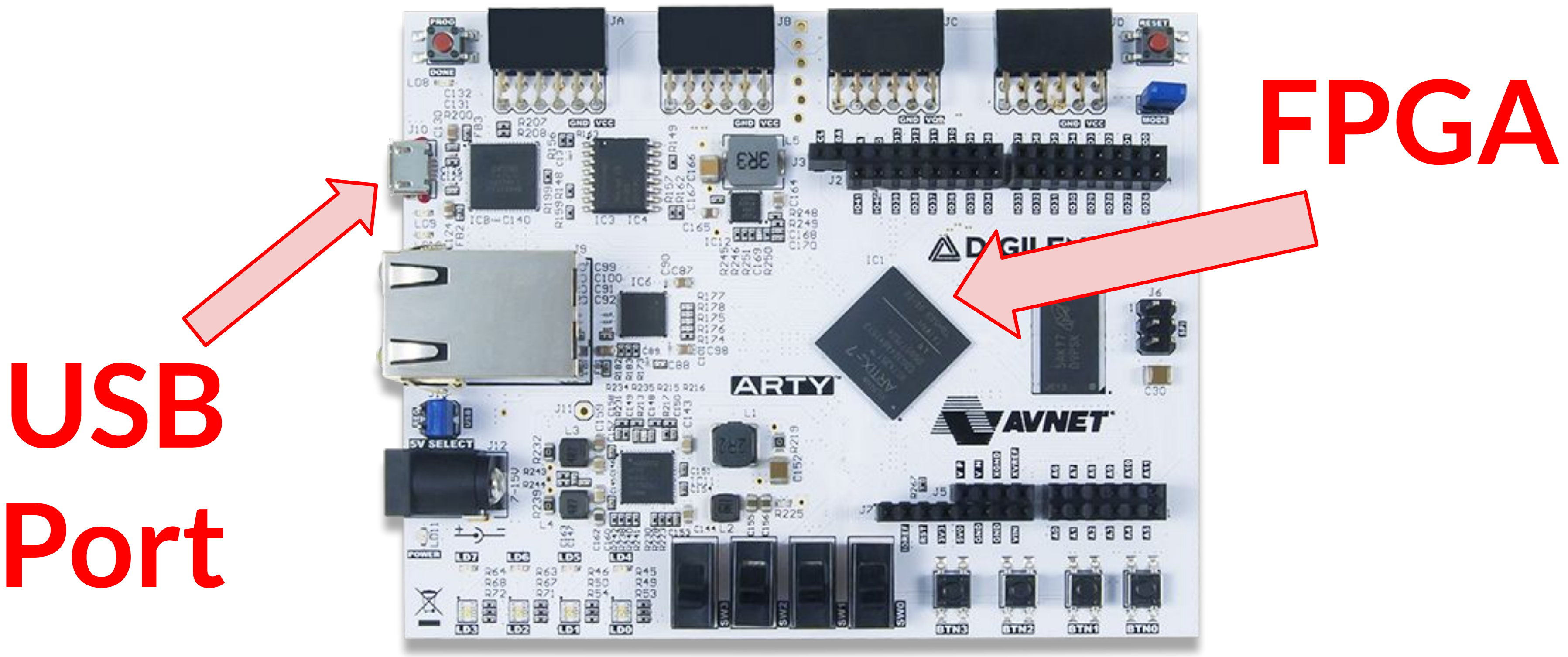
Disclaimer: NOT an official Google project

It's on GitHub: Fork it, fix it, send a PR!

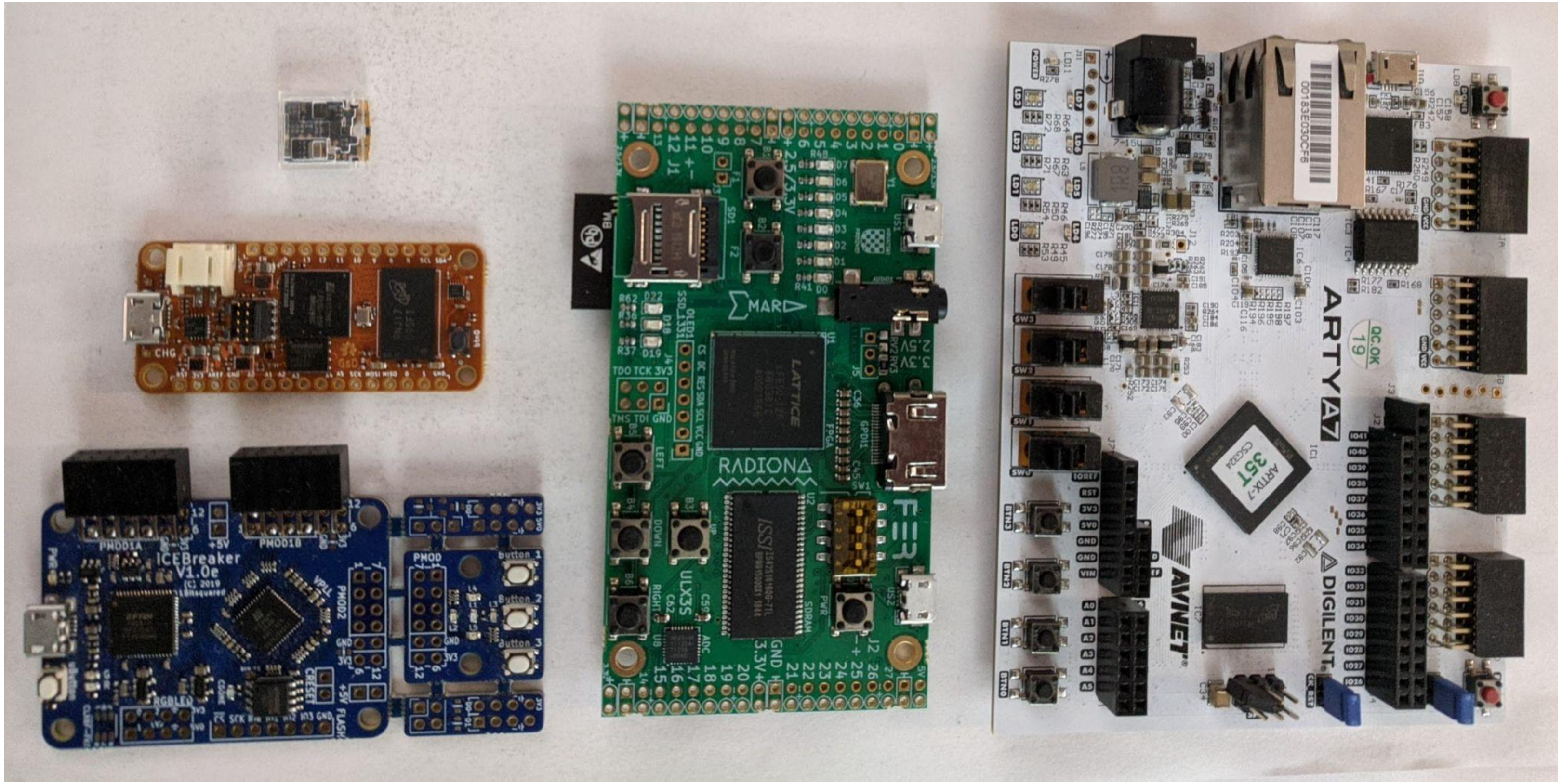
The CFU Playground

For HW & SW engineers building

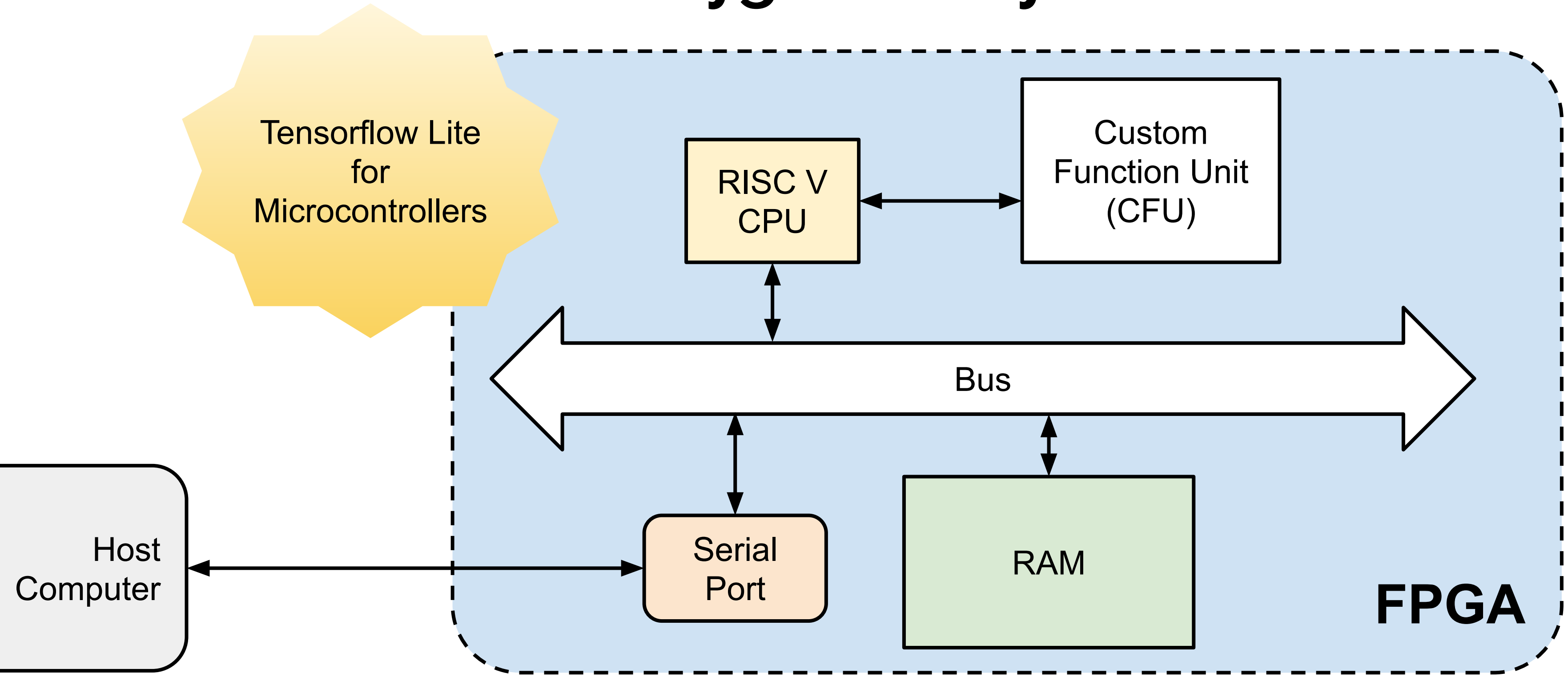
- fast, low-power ML accelerators
- on a completely open stack
- with FPGAs
- with quick design iterations


FPGA
USB Port
ARTY
DIGILENT
AVNET
ARTIX-7

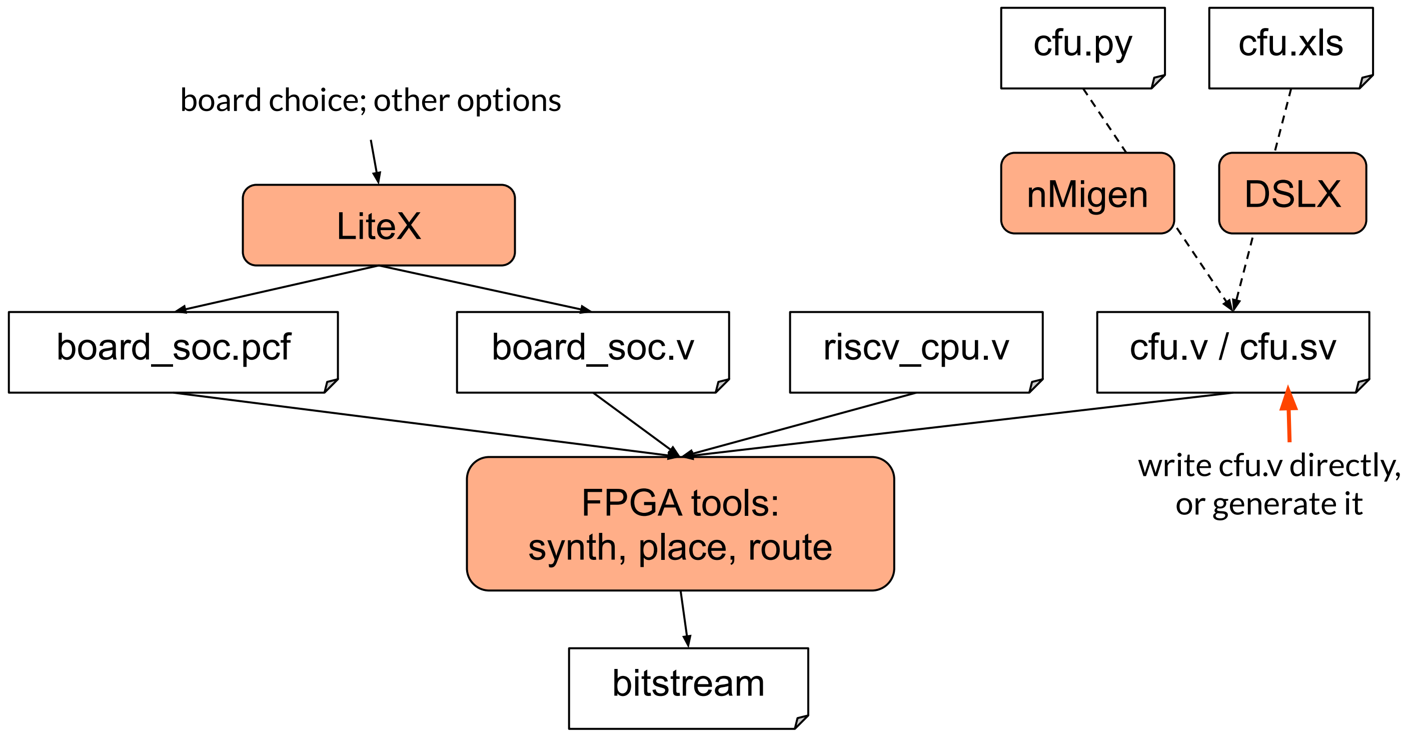
CFU Playground -- some of the tested boards



CFU Playground System Architecture


Tensorflow Lite for Microcontrollers
RISC V CPU
Custom Function Unit (CFU)
Bus
Host Computer
Serial Port
RAM
FPGA

Hardware Flow



cfu.py
cfu.xls
board choice; other options
nMigen
DSLX
LiteX
board_soc.pcf
board_soc.v
riscv_cpu.v
cfu.v / cfu.sv
write cfu.v directly, or generate it
FPGA tools: synth, place, route
bitstream

Open Source Showcase!

● ML library	TensorFlow	-- *open source*
● CPU ISA	RISC-V	-- *open*
● CPU design	VexRiscv	-- *open source*
● FPGA SoC/IP	LiteX	-- *open source*
● FPGA synth/PnR	SymbiFlow,Yosys, Nextpnr, VPR	-- *open source*
○ FPGA vendor tools can be used if you wish		
● Python HW gen	Migen, nMigen	-- *open source*
● Simulation	Renode, Verilator	-- *open source*

- The only proprietary component is the FPGA itself

Benefits of Open Source

- No licensing fees, contracts, EULAs
- No vendor lock-in
- Transparency -- you can inspect all of the RTL
- Entire flow in-house

CFU Playground Uses

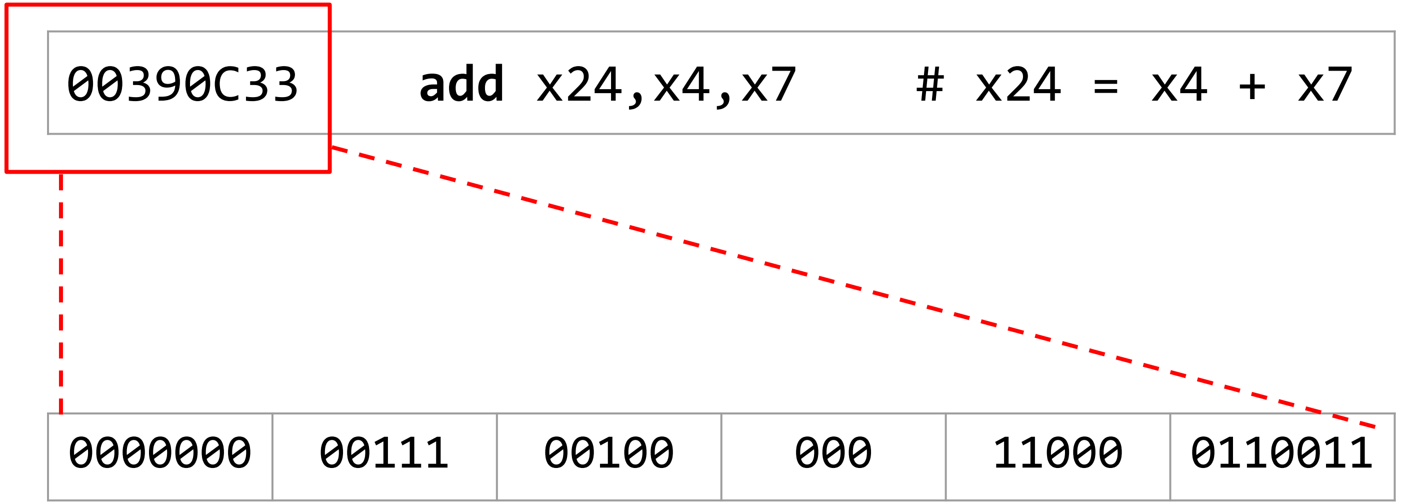
- **Deploy** a soft CPU+CFU for tinyML
    - Updates include new ML model, software, and CPU+CFU
    - Lower risk than many alternatives
- **Prototype** a custom RISC-V-based ASIC
- **Learn** about ML software, hardware, and performance
- **Research** new ML approaches
    - while co-designing the hardware to support it

CFU Basics

The RISCV Add Instruction

```
00390C33    add x24,x4,x7    # x24 = x4 + x7
```

The RISCV Add Instruction



```
00390C33     add x24,x4,x7     # x24 = x4 + x7
```

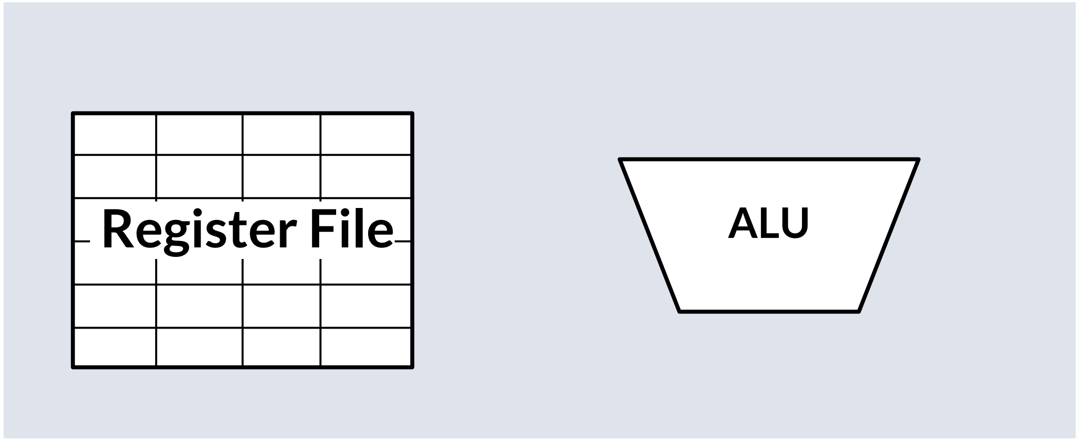
0000000	00111	00100	000	11000	0110011

The RISCV Add Instruction

```
00390C33     add x24,x4,x7     # x24 = x4 + x7
```

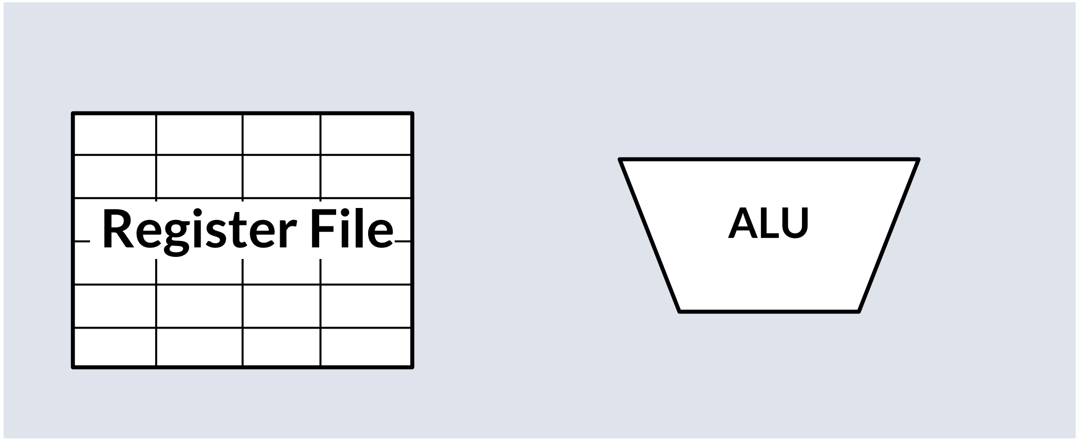
0000000	00111	00100	000	11000	0110011
					opcode
					ALU

The RISCV Add Instruction



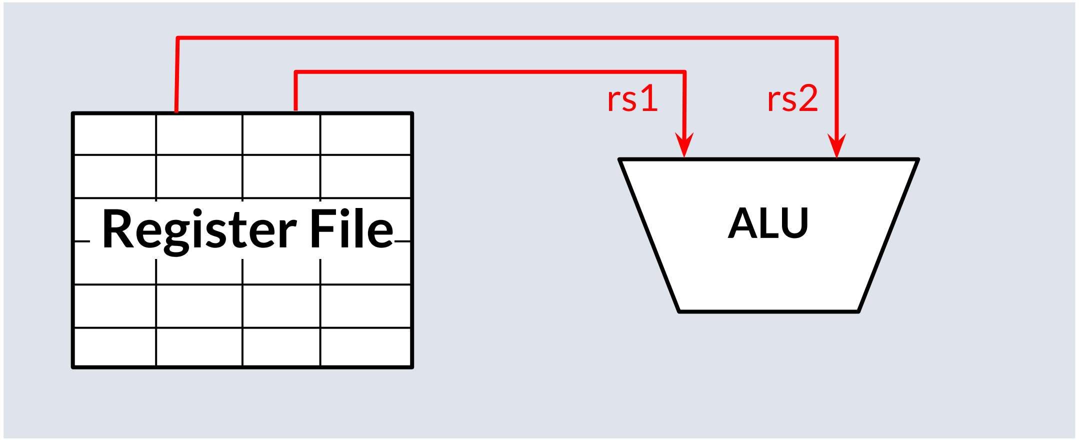
0000000	00111	00100	000	11000	0110011
					opcode
					ALU

The RISCV Add Instruction



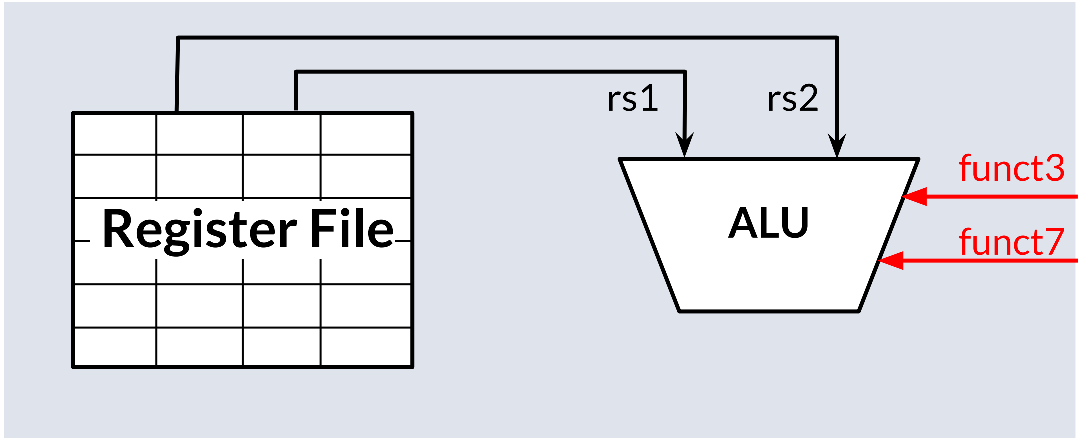
0000000	00111	00100	000	11000	0110011
funct7	rs2	rs1	funct3	rd	opcode
ADD	x7	x4	ADD	x24	ALU

The RISCV Add Instruction



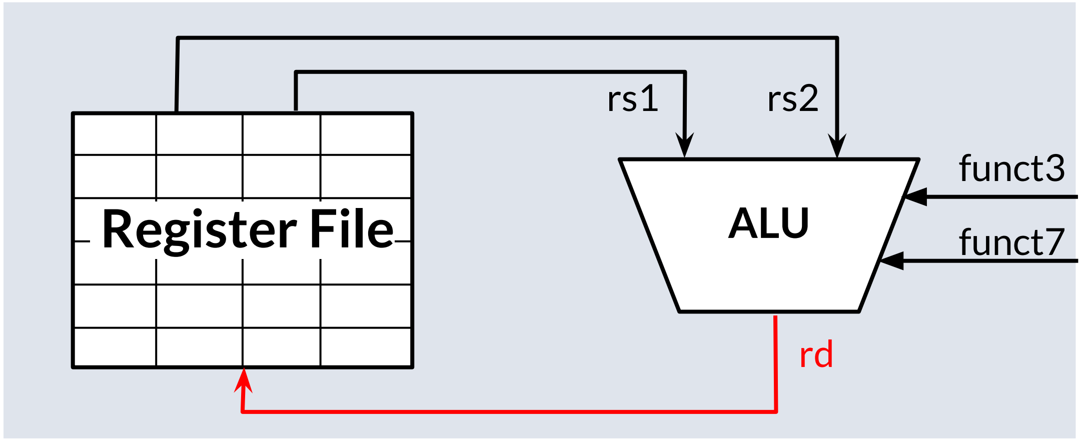
0000000	00111	00100	000	11000	0110011
funct7	rs2	rs1	funct3	rd	opcode
ADD	x7	x4	ADD	x24	ALU

The RISCV Add Instruction



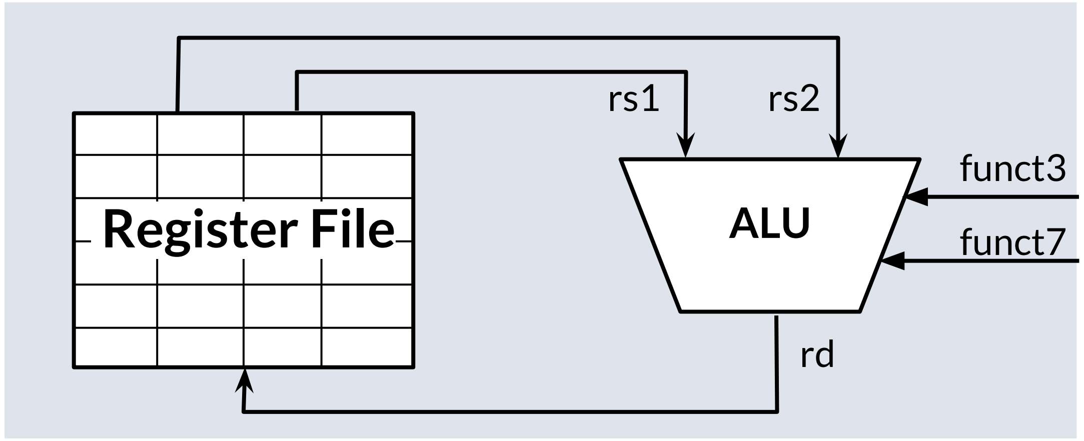
0000000	00111	00100	000	11000	0110011
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
ADD	x7	x4	**ADD**	x24	**ALU**

The RISCV Add Instruction



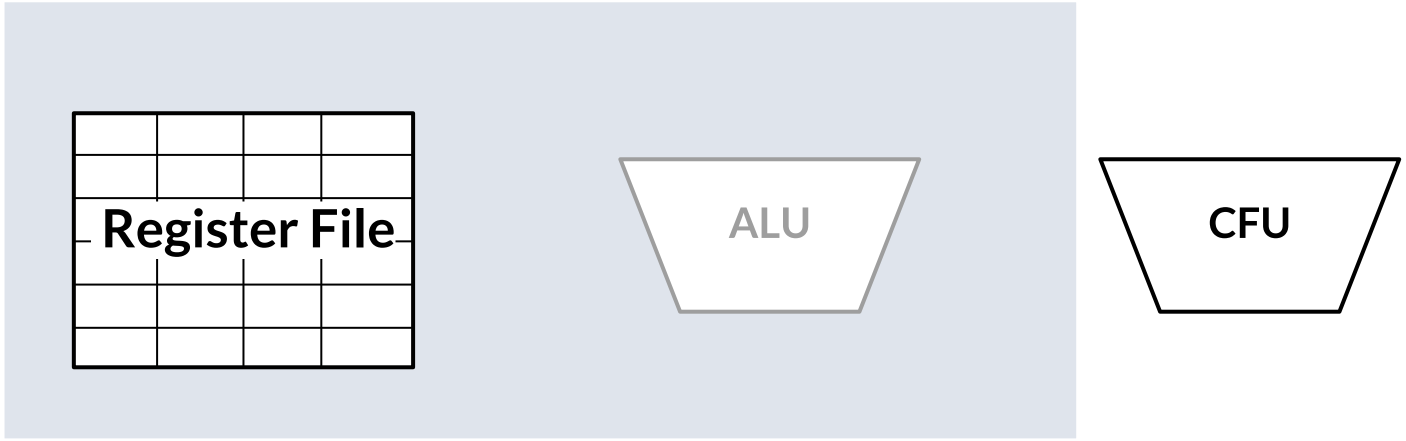
0000000	00111	00100	000	11000	0110011
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
ADD	x7	x4	**ADD**	x24	**ALU**

The RISCV Add Instruction



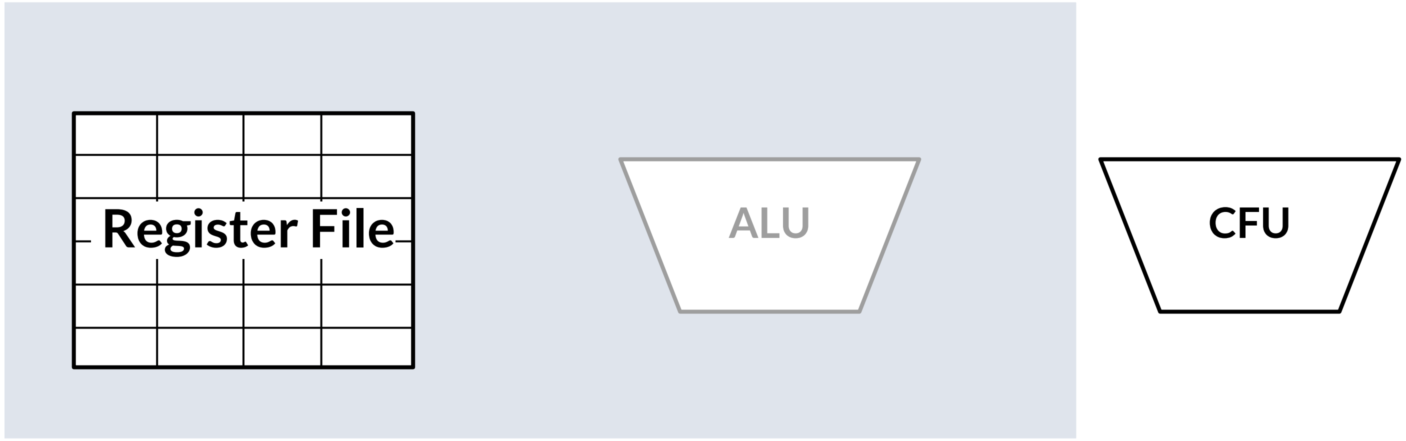
0000000	00111	00100	000	11000	0110011
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
ADD	x7	x4	**ADD**	x24	**ALU**

The RISCV Custom Instruction



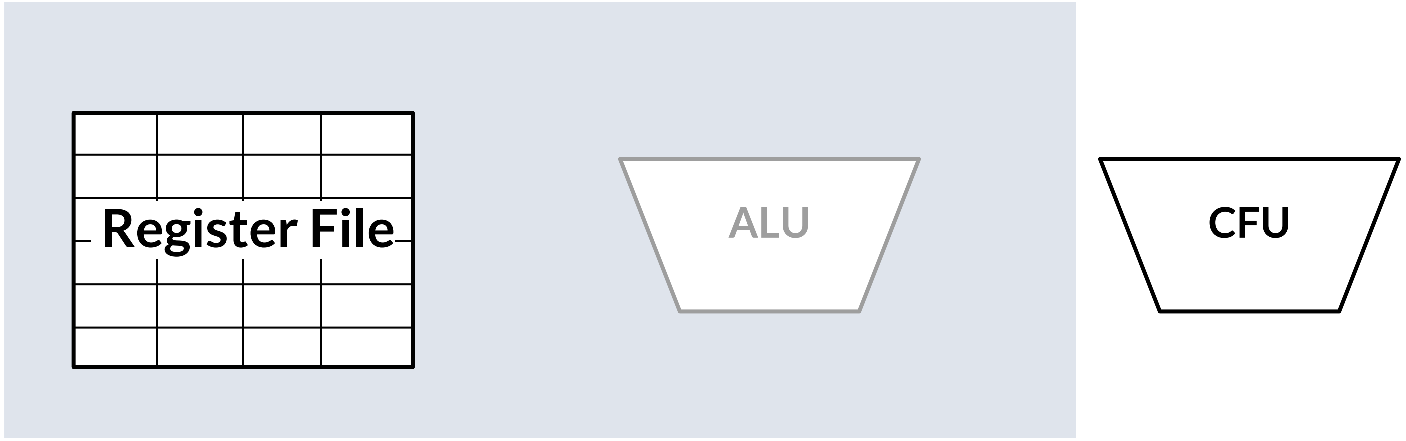
0000000	00111	00100	000	11000	0001011
					opcode

The RISCV Custom Instruction



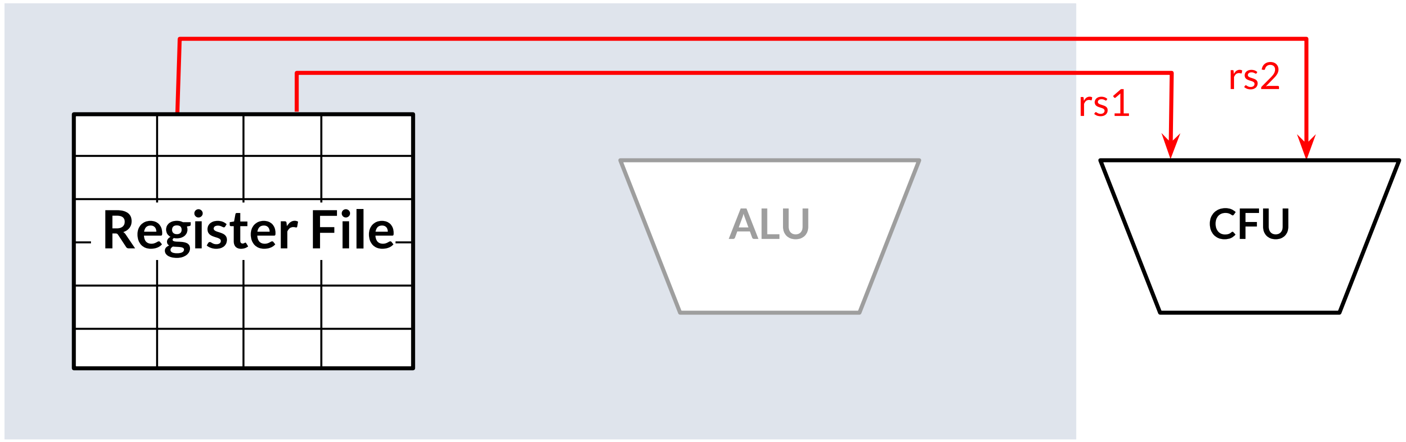
0000000	00111	00100	000	11000	0001011
					opcode
					CUSTOM

The RISCV Custom Instruction



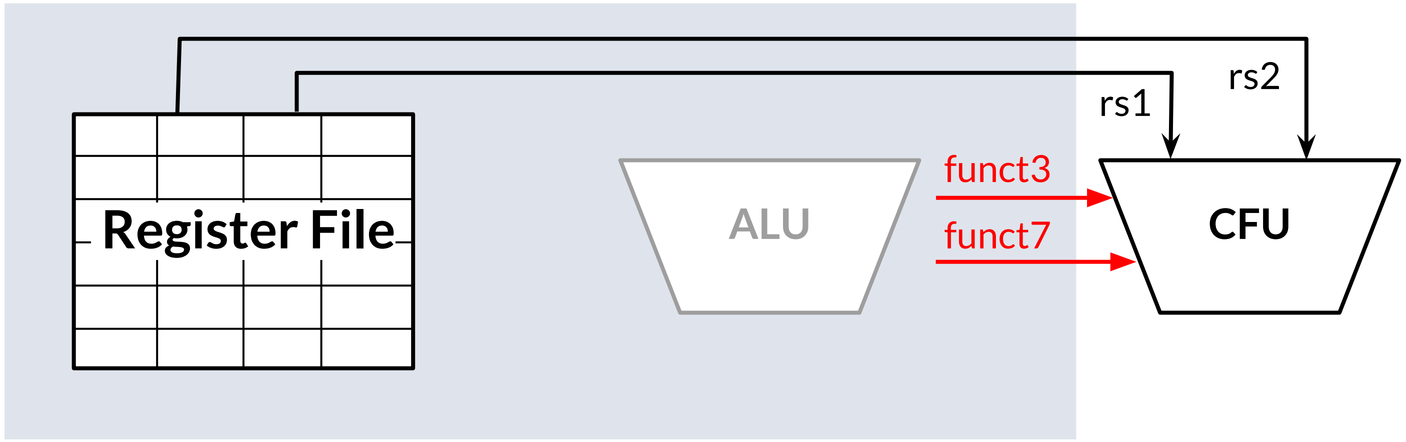
0000000	00111	00100	000	11000	0001011
funct7	rs2	rs1	funct3	rd	opcode
op	x7	x4	op	x24	CUSTOM

The RISCV Custom Instruction



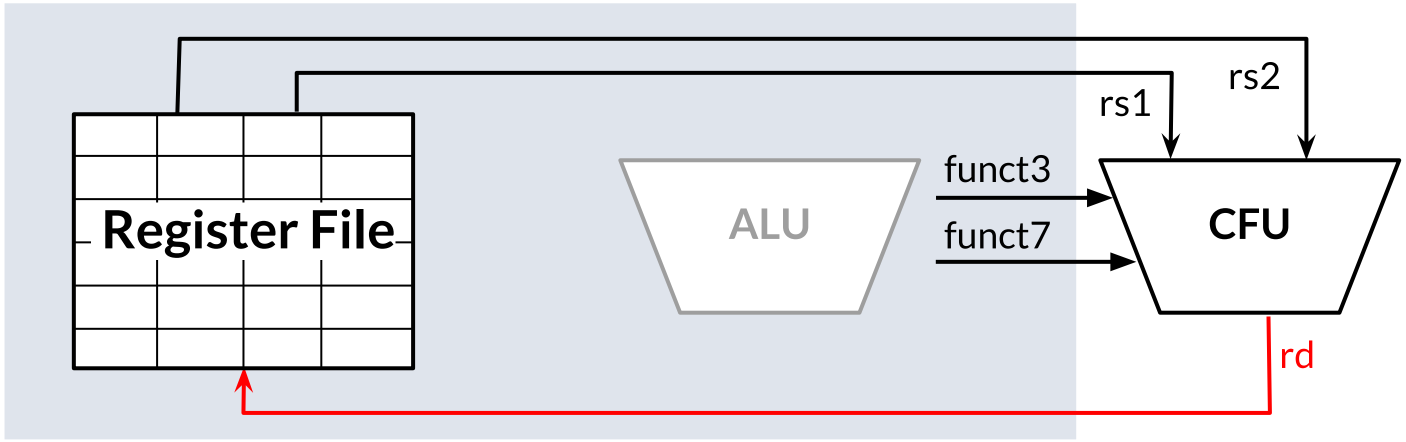
0000000	00111	00100	000	11000	0001011
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
op	x7	x4	**op**	x24	**CUSTOM**

The RISCV Custom Instruction



0000000	00111	00100	000	11000	0001011
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
op	x7	x4	**op**	x24	**CUSTOM**

The RISCV Custom Instruction



0000000	00111	00100	000	11000	0001011
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
op	x7	x4	**op**	x24	**CUSTOM**

Using CFU ops from C++

- cfu_op macro expands to GCC inline asm

```
rslt = cfu_op(funct3, funct7, op1, op2);
```

Compile-time constants

C / C++ variables / expressions

```
t1 = *x;
t2 = cfu_op(0, 0, t1, b);
t3 = cfu_op(1, 0, t2, b);
*x = t3;
```

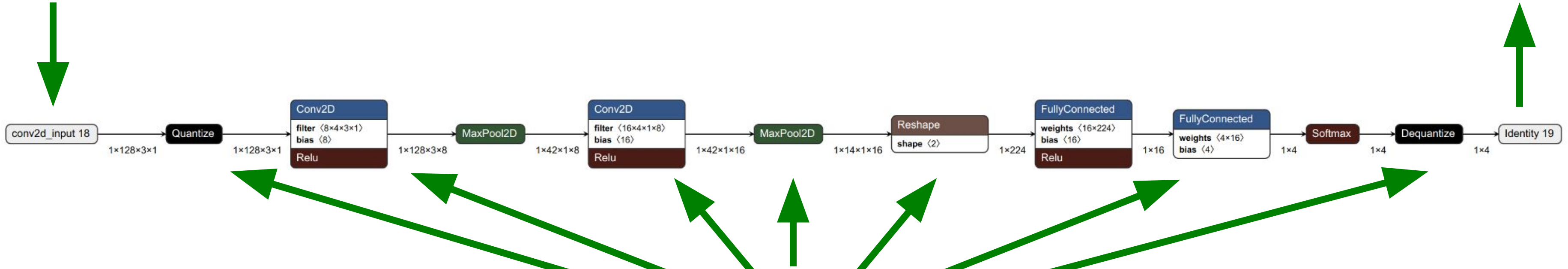
```
400001a0:   00812783    lw         a5,8(sp)
400001a4:   00d7878b    cfu[0,0]   a5, a5, a3
400001a8:   00d7978b    cfu[0,1]   a5, a5, a3
400001ac:   00f12423    sw         a5,8(sp)
```

A model is a series of operations

not hotdog

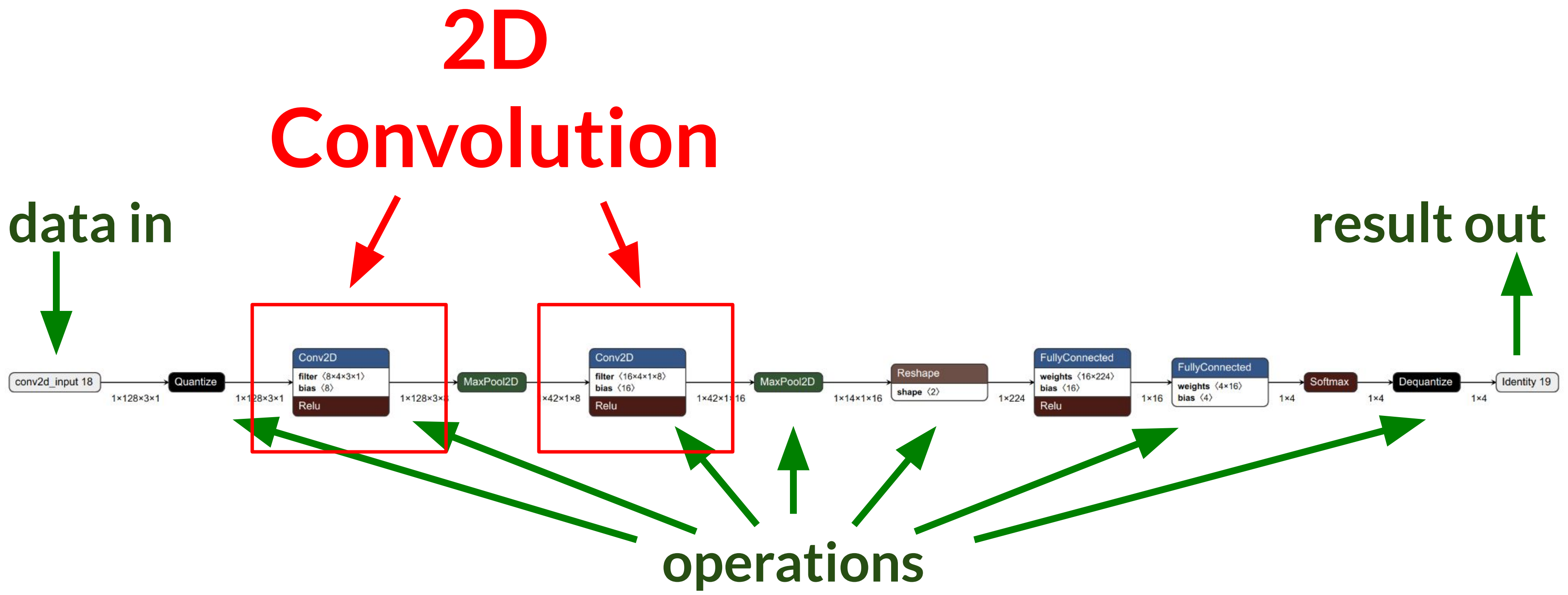
data in

result out



operations

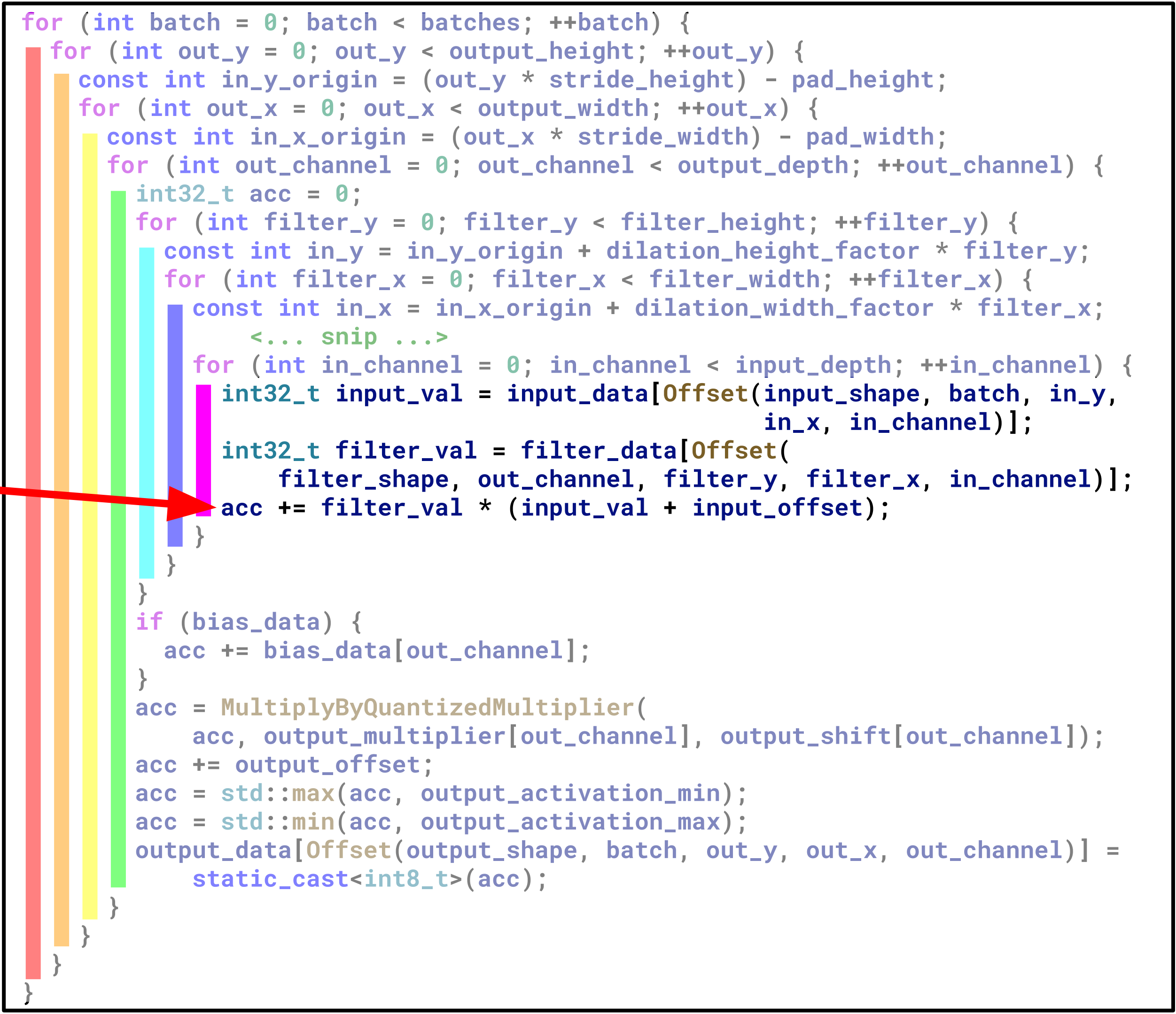
A model is a series of operations



2D Convolution

Loops nested 7(!) levels

Inner loop is simple
- fetches input
- fetches filter
- acc += X * (Y+Z)



```
for (int batch = 0; batch < batches; ++batch) {
  for (int out_y = 0; out_y < output_height; ++out_y) {
    const int in_y_origin = (out_y * stride_height) - pad_height;
    for (int out_x = 0; out_x < output_width; ++out_x) {
      const int in_x_origin = (out_x * stride_width) - pad_width;
      for (int out_channel = 0; out_channel < output_depth; ++out_channel) {
        int32_t acc = 0;
        for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
          const int in_y = in_y_origin + dilation_height_factor * filter_y;
          for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
            const int in_x = in_x_origin + dilation_width_factor * filter_x;
                <... snip ...>
            for (int in_channel = 0; in_channel < input_depth; ++in_channel) {
              int32_t input_val = input_data[Offset(input_shape, batch, in_y,
                                                    in_x, in_channel)];
              int32_t filter_val = filter_data[Offset(
                  filter_shape, out_channel, filter_y, filter_x, in_channel)];
              acc += filter_val * (input_val + input_offset);
            }
          }
        }
        if (bias_data) {
          acc += bias_data[out_channel];
        }
        acc = MultiplyByQuantizedMultiplier(
            acc, output_multiplier[out_channel], output_shift[out_channel]);
        acc += output_offset;
        acc = std::max(acc, output_activation_min);
        acc = std::min(acc, output_activation_max);
        output_data[Offset(output_shape, batch, out_y, out_x, out_channel)] =
            static_cast<int8_t>(acc);
      }
    }
  }
}
```

2D Convolution

Running an ML model is

acc += X * (Y+Z)

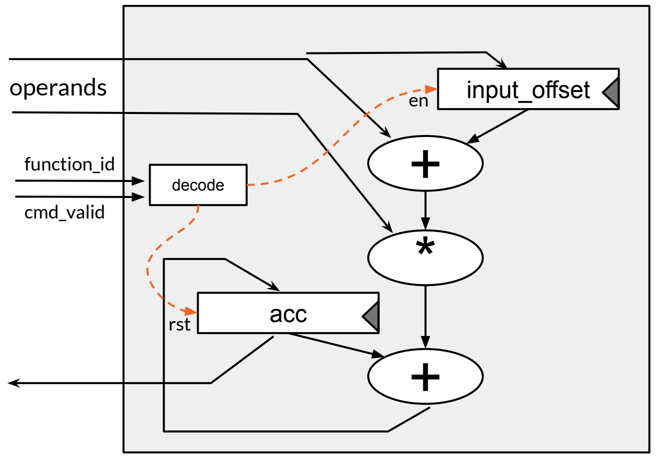
... many times

```
for (int batch = 0; batch < batches; ++batch) {
  for (int out_y = 0; out_y < output_height; ++out_y) {
    const int in_y_origin = (out_y * stride_height) - pad_height;
    for (int out_x = 0; out_x < output_width; ++out_x) {
      const int in_x_origin = (out_x * stride_width) - pad_width;
      for (int out_channel = 0; out_channel < output_depth; ++out_channel) {
        int32_t acc = 0;
        for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
          const int in_y = in_y_origin + dilation_height_factor * filter_y;
          for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
            const int in_x = in_x_origin + dilation_width_factor * filter_x;
                <... snip ...>
            for (int in_channel = 0; in_channel < input_depth; ++in_channel) {
              int32_t input_val = input_data[Offset(input_shape, batch, in_y,
                                                    in_x, in_channel)];
              int32_t filter_val = filter_data[Offset(
                  filter_shape, out_channel, filter_y, filter_x, in_channel)];
              acc += filter_val * (input_val + input_offset);
            }
          }
        }
        if (bias_data) {
          acc += bias_data[out_channel];
        }
        acc = MultiplyByQuantizedMultiplier(
            acc, output_multiplier[out_channel], output_shift[out_channel]);
        acc += output_offset;
        acc = std::max(acc, output_activation_min);
        acc = std::min(acc, output_activation_max);
        output_data[Offset(output_shape, batch, out_y, out_x, out_channel)] =
            static_cast<int8_t>(acc);
      }
    }
  }
}
```

Typical CFU evolution

Loop invariant

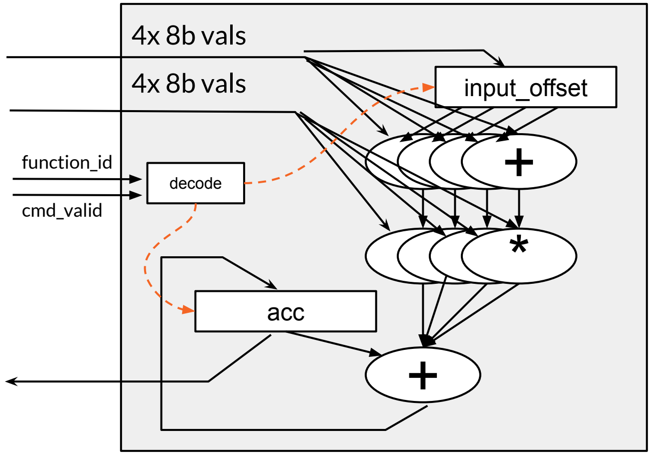
acc += filter_val * (input_val + input_offset)



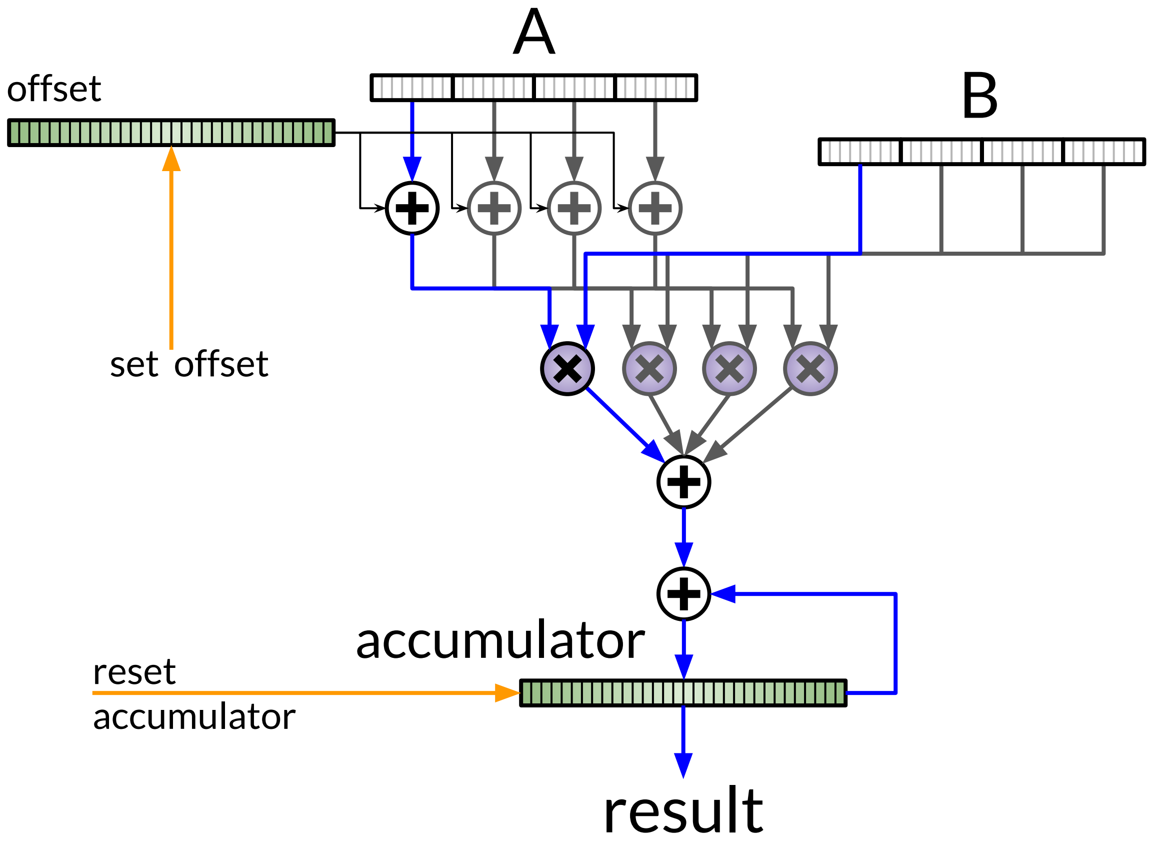
Typical CFU evolution

Loop invariant

acc += filter_val * (input_val + input_offset)



Exploit SIMD


A
B
offset
set offset
accumulator
reset
accumulator
result

CFU Playground Build

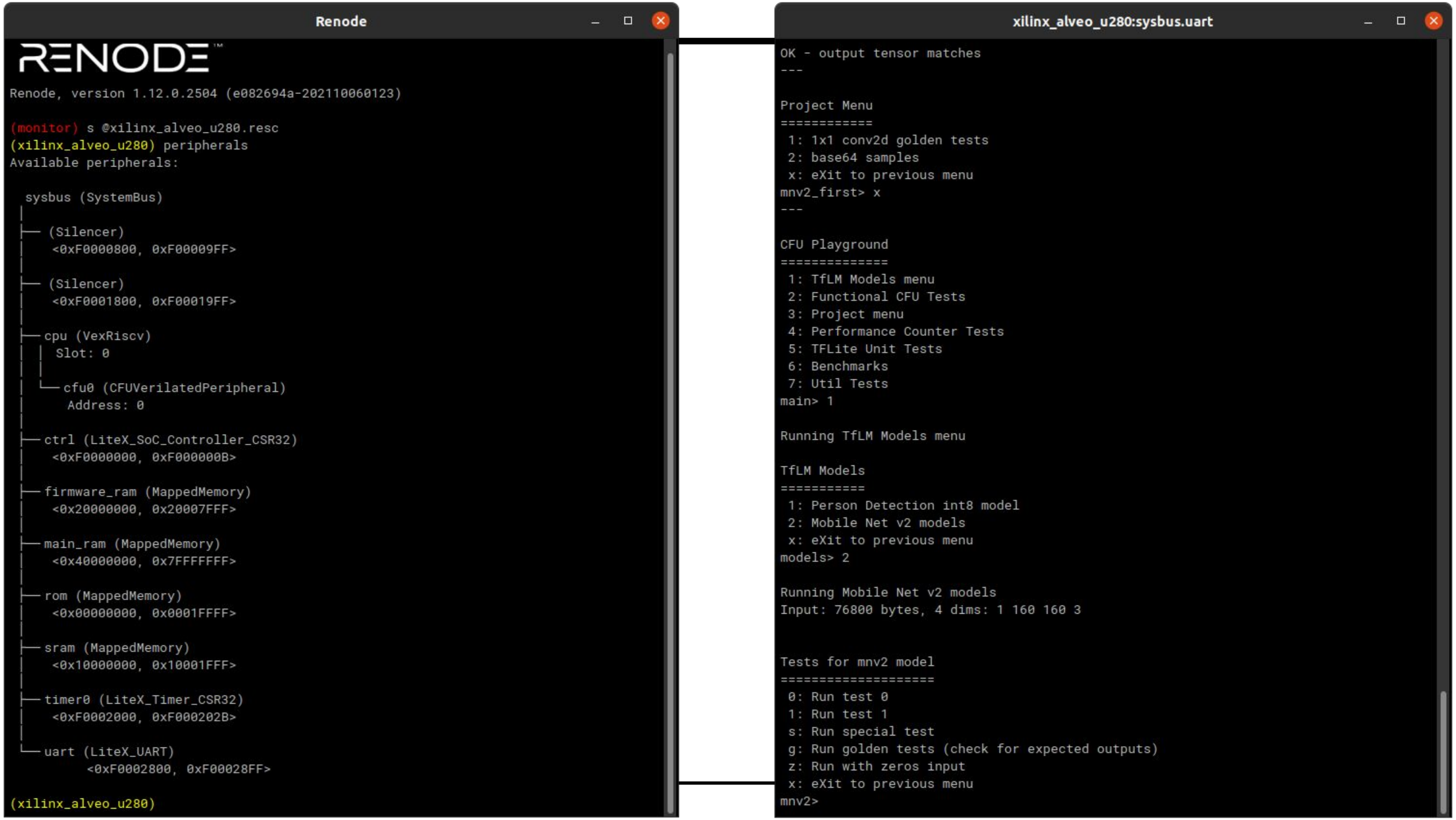
```
$ cd proj/proj_template_v/        # establish project

$ make prog           3.5min      # build gateware using ./cfu.v
                                    and put the bitstream on the FPGA

$ make -j8 software   <1min       # build software w/ local overrides

$ make load                       # load and execute the software
```

...then interact with the software running on the board

DEMO

Simulation / Debug

- Renode + Verilator
  - Recent work by Antmicro
  - Can accurately model a board you don’t have
  - ISA simulation of the RISC-V
    - Fast
    - Can attach GDB
  - Verilog-level simulation of the CFU using Verilator plugin
    - Can capture waveforms
  - Not cycle accurate



```
Renode, version 1.12.0.2504 (e082694a-202110060123)

(monitor) s @xilinx_alveo_u280.resc
(xilinx_alveo_u280) peripherals
Available peripherals:

  sysbus (SystemBus)
  │
  ├── (Silencer)
  │    <0xF0000800, 0xF00009FF>
  │
  ├── (Silencer)
  │    <0xF0001800, 0xF00019FF>
  │
  ├──cpu (VexRiscv)
  │  │ Slot: 0
  │  │
  │  └──cfu0 (CFUVerilatedPeripheral)
  │       Address: 0
  │
  ├──ctrl (LiteX_SoC_Controller_CSR32)
  │    <0xF0000000, 0xF000000B>
  │
  ├──firmware_ram (MappedMemory)
  │    <0x20000000, 0x20007FFF>
  │
  ├──main_ram (MappedMemory)
  │    <0x40000000, 0x7FFFFFFF>
  │
  ├──rom (MappedMemory)
  │    <0x00000000, 0x0001FFFF>
  │
  ├──sram (MappedMemory)
  │    <0x10000000, 0x10001FFF>
  │
  ├──timer0 (LiteX_Timer_CSR32)
  │    <0xF0002000, 0xF000202B>
  │
  └──uart (LiteX_UART)
         <0xF0002800, 0xF00028FF>

(xilinx_alveo_u280)
```

```
OK - output tensor matches
---

Project Menu
============
 1: 1x1 conv2d golden tests
 2: base64 samples
 x: eXit to previous menu
mnv2_first> x
---

CFU Playground
==============
 1: TfLM Models menu
 2: Functional CFU Tests
 3: Project menu
 4: Performance Counter Tests
 5: TFLite Unit Tests
 6: Benchmarks
 7: Util Tests
main> 1

Running TfLM Models menu

TfLM Models
===========
 1: Person Detection int8 model
 2: Mobile Net v2 models
 x: eXit to previous menu
models> 2

Running Mobile Net v2 models
Input: 76800 bytes, 4 dims: 1 160 160 3


Tests for mnv2 model
====================
 0: Run test 0
 1: Run test 1
 s: Run special test
 g: Run golden tests (check for expected outputs)
 z: Run with zeros input
 x: eXit to previous menu
mnv2>
```

How to Have Fun with CFU-Playground

Try the tutorials: cfu-playground.readthedocs.io

Get the source: github.com/google/CFU-Playground

Send bugs and patches

Contact us!

- mail (tcal@google.com)
- we hang out on #litex and #symbiflow at libera.chat
- raise an issue at github.com/google/CFU-Playground

Contribute!

- Try a new FPGA board
- File issues (including documentation)
- Add a demo
- Add a new TFLite model
- Design your own CFU for ML or non-ML!

End

FAQ

- Can I run a demo?
  - We didn’t emphasize interactive demos. Here we’re focused on performance and easy development. But, it should be straightforward to export your CFU + modified TFLM code to a demo.
- Isn’t the CPU the bottleneck? Can I add DMA?
  - Yes, not currently, but we’re thinking about it.
- Can I use my specific FPGA board?
  - It will be easy if it’s a LiteX-supported board. If it is, just try it by adding TARGET=<board_name>. If it *isn’t* a LiteX-supported board, start by adding it to LiteX at github.com/litex-hub/litex-boards.

FAQ

- Do I need to use VexRiscv?
  - Currently VexRiscv is the only soft CPU that has this specific CFU interface. We would love to work with other CPU core authors to add the CFU interface to their CPUs.
- Do you support I-format instructions?
  - It is moderately easy to rebuild the VexRiscv to support I-format instructions. We had support for them, but weren't using them, and observed that supporting them made it more difficult to meet timing constraints, so currently the VexRiscv being used does *not* support I-format CFU instructions.